CPSC 526 - Assignment 5

Due date: Thursday, November 23, 2017 at 23:59.
Weight: 16% of your final grade.
Group work allowed, maximum group size is 2 people.

In this assignment you are going to write a personal firewall simulator for IPv4 packets. The simulator will be capable of filtering both incoming and outgoing packets. Please note that there will be no networking part to this assignment at all. The program will simply read a specified configuration file, then read lines from standard input and output results to standard output.

When the simulator starts, it will first read in the firewall rules from a configuration file. The name of the configuration file will be specified on the command line. After reading in the rules, the simulator will then start reading standard input line by line. Each line on the standard input will describe a simulated packet. For each packet the program will determine what to do with the packet based on the rules from the configuration file, and print information about its decision to standard output.

Configuration file format

The configuration file contains an ordered list of firewall rules, one rule per line. Note that the order of the rules is important, as was discussed during lectures. Each line has 4 mandatory fields and one optional field:

```
<direction> <action> <ip> <port> [flag]
```

The fields will be separated by one or more white spaces (space or TAB). It is also possible that each line will have trailing and leading white spaces. The fields are explained below:

<direction>	Specifies the direction of traffic to which this rule applies. Allowed values are "in" and "out".
<action>	Specifies the action to be taken when the packet matches this rule. Allowed values are "accept", "drop" and "reject".
<ip>	Defines the IP range for this rule. There are two different ways to specify a range: • using CIDR notation, e.g. `136.159.22.0/24`, or • using "*" to match any address.
<ports>	Defines the list of destination ports for which this rule will apply. Each port is an integer between 0-65535. The list can contain one or more ports separated by commas. It is also possible to specify "any" port by using the wildcard "*".
[flag]	This optional field describes whether the rule will be applied only to packets that are part of an established connection or to all packets. If the field is not present, the rule will apply to all packets. If present, the rule will only apply to established packets. If present, the only allowed value is "established".

The configuration file may also contain comment lines and empty lines, which should be ignored. Comment lines are lines where the first non-white space character is a hash "#". There is no need for you to support trailing comments, although you may do so if you wish. Empty line is any line that has either no characters, or only whitespace characters. Example of a configuration file is below:

```
in  accept 136.159.0.0/16  22,80
in  accept *               *       established
in  reject *               *
out accept *               *       established
```

The rules in the example above would result in the following behavior:

- any incoming packet from IP addresses 136.159.*.* to ports 22 or 80 will be accepted
- any incoming established packets will be accepted, from any IP to any port
- any other incoming packets would be rejected
- all outgoing established packets will be accepted

If the simulator cannot find a rule for a packet, it will report the default action, which is "drop".

Packet format

The packets will be read in from standard input, one packet per line. Your program should read the standard input until EOF, at which point it should terminate. Each packet will have exactly 4 fields:

```
<direction> <ip> <port> <flag>
```

The fields will be separated by one or more white spaces (space or TAB). Any line could also contain trailing and leading white spaces. You do not need to support comments and empty lines for the packets that you read in from standard input, although you can if you wish. The fields are explained below:

<direction>	Specifies the direction of the packet. Each packet is either incoming or outgoing. The only allowed values are “in” and “out”.
<ip>	For incoming packets this specifies the source IP address. For outgoing packets this specifies the destination IP address. In either case, the address will be specified in dot-decimal notation, e.g. 136.159.5.22.
<port>	This specifies the destination port of the packet, and it is an integer between 0-65535.
<flag>	Boolean flag (0 or 1) specifying whether the packet is part of a new (0) session or established (1) session.

Example with 3 packets:

```
in 136.159.5.22 22 0
in 10.0.0.44 443 1
out 10.0.1.1 80 0
```

The 3 packets above are:

- The 1st packet is coming from 136.159.5.22 and its destination is port 22 (SSH). It is a new packet.
- The 2nd packet is coming from 10.0.0.44 and its destination is port 443 (HTTPS). This packet is part of an existing session.
- The 3rd packet is an outgoing packet, going to 10.0.1.1 to port 80. It is a new packet.

Output

Note: Marking of your program will most likely involve some automation so it is very important that you follow the output format precisely.

For each packet the simulator will output exactly one line to standard output, and nothing else. The format of the output is:

```
<action>(<rule line number>) <direction> <ip> <port> <flag>
```

The meanings of the fields are:

<action>	The action that the firewall decided should be taken. If no rule could be found for a packet, the action should be “drop”.
<rule line number>	The line number where the rule responsible for this action can be found. The lines are numbered starting from 1. If no rule could be found for a packet, the line number should be omitted. Note that all lines in the configuration file are numbered, including empty lines and comment lines.
<direction> **<ip>** **<port>** **<flag>**	The fields of the packet.

Example output:

```
accept(1) in 136.159.5.22 22 0
reject(2) in 10.0.0.44 443 1
drop() out 10.0.1.1 80 0
```

The above output means:

- the first packet was accepted by a rule on line #1
- the second packet was denied by a rule on line #2
- the third packet was dropped by default (no rule was found that would match the packet).

Full example

A complete example, with configuration file, input and corresponding output is below:

Configuration file (firewall rules)

```
in  accept 136.159.5.5/32     22
in  accept 136.159.5.5/16     80,8080
in  accept *                  443
in  accept 10.0.0.0/31        *
in  reject *                  21
in  accept *                  *        established
out accept 137.159.0.0/8      *
out reject 10.0.0.0/8         *
out reject *                  22
out accept *                  *
```

Input packets:	Program output:
in 136.159.5.5 22 0	accept(1) in 136.159.5.5 22 0
in 136.159.5.5 23 0	drop() in 136.159.5.5 23 0
in 136.159.5.6 22 1	accept(6) in 136.159.5.6 22 1
in 136.159.255.5 80 0	accept(2) in 136.159.255.5 80 0
in 136.159.0.5 8080 1	accept(2) in 136.159.0.5 8080 1
in 136.159.255.0 8080 0	accept(2) in 136.159.255.0 8080 0
in 136.158.5.5 8080 0	drop() in 136.158.5.5 8080 0
in 24.25.26.27 443 0	accept(3) in 24.25.26.27 443 0
in 24.25.26.27 444 0	drop() in 24.25.26.27 444 0
in 24.25.26.27 444 1	accept(6) in 24.25.26.27 444 1
in 24.25.26.27 21 1	reject(5) in 24.25.26.27 21 1
in 10.0.0.0 1000 0	accept(4) in 10.0.0.0 1000 0
in 10.0.0.1 1000 0	accept(4) in 10.0.0.1 1000 0
in 10.0.0.2 1000 0	drop() in 10.0.0.2 1000 0
out 137.255.0.255 33 0	accept(7) out 137.255.0.255 33 0
out 10.0.0.133 1 0	reject(8) out 10.0.0.133 1 0
out 5.5.5.5 22 0	reject(9) out 5.5.5.5 22 0
out 5.5.5.5 22 1	reject(9) out 5.5.5.5 22 1
out 5.5.5.5 23 1	accept(10) out 5.5.5.5 23 1

Malformed input

You need to handle malformed input in a sensible way (both rules and packets). If you detect an input that is invalid, I expect you to handle it in one of two ways:

- you report an error and your program stops running; or
- you report a warning and ignore the offending line, but your program continues to run.

What you are not allowed to do is to let your program crash under any circumstances, no matter how bad the input is.

Additional notes

- You may implement your program in Python, C or C++.

- Your program must run on the Linux machines in the labs (MS).
- If you implement your program in python, your main module must be called '`fw.py`'. If you implement your program in C/C++ the executable must be called '`fw`'.
- You may **not** call any external programs, and you may **not** use any libraries, such as for parsing CIDR notation, IP address parsing, or IP address matching. You must implement all this functionality yourself.
- You will be required to demo your assignment, and during the demo you will be asked to demonstrate your familiarity with all of the code. If you do decide to work on the assignment in a group, both of you should understand the code 100%.

Submission

You need to submit your source code and a **`readme.pdf`** file to D2L. Please use ZIP or TAR archives. If you decide to work in a group, both group members need to submit the assignment. The **`readme.pdf`** file must include:

- your name, ID and tutorial section;
- name of your group partner if applicable;
- a section that describes how to compile/run your code.

You must submit the above to D2L to receive any marks for this assignment.

Marking

Marking will consist of running your program against a fixed set of inputs (configuration files and packets). Each test case will be marked based on the number of correct output lines. Your assignment mark will be determined based on the total score. Here is a possible way your assignment will be tested:

```
$ python3 fw.py rules1.txt < packets1.txt | diff -y results1.txt -
```

The above line runs your python code `fw.py` with rules defined in `rules1.txt` and packets defined in file `packets1.txt`, then compares the output of your program with the correct output in `results1.txt`. If time permits, I will post sample inputs and outputs in the next few days for you to be able to start testing your programs.

Please format and document your source code. Ugly/undocumented source code will be subject to 50% penalty.

General information about all assignments:

1. Late assignments or components of assignments will not be accepted for marking without approval for an extension beforehand. What you have submitted in D2L as of the due date is what will be marked.
2. Extensions may be granted for reasonable cases, but only by the course instructor, and only with the receipt of the appropriate documentation (e.g., a doctor's note). Typical examples of reasonable cases for an extension include: illness or a death in the family. Cases where extensions will not be granted include situations that are typical of student life, such as having multiple due dates, work commitments, etc. Forgetting to hand in your assignment on time is not a valid reason for getting an extension.
3. After you submit your work to D2L, make sure that you check the content of your submission. It's your responsibility to do this, so make sure that your submit your assignment with enough time before it is due so that you can double-check your upload, and possibly re-upload the assignment.
4. All assignments should include contact information, including full name, student ID and tutorial section, at the very top of each file submitted.
5. Although group work is allowed, you are not allowed to copy the work of others. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.
6. You can and should submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It is better to submit incomplete work for a chance of getting partial marks, than not to submit anything.
7. Only one file can be submitted per assignment. If you need to submit multiple files, you can put them into a single container. Supported container types are TAR or gzipped TAR. No other formats will be accepted.
8. Assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have question after you have talked to your TA then you can contact your instructor.